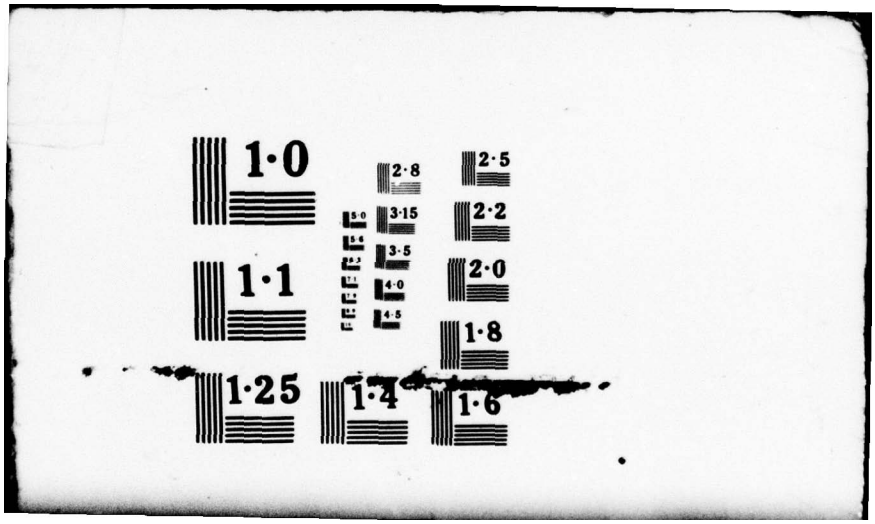
AD-A067 754 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/G 12/2
THE SIMPLEX SON ALGORITHM FOR LP/EMBEDDED NETWORK PROBLEMS.(U)
JAN 79 F GLOVER, D KLINGMAN N00014-75-C-0616
UNCLASSIFIED CCS-317 NL

| OF |
ADA
067754

END
DATE
FILMED
6-79
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6

LEVEL II

12

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

(12)

Research Report CCS 317

THE SIMPLEX SON ALGORITHM FOR LP/EMBEDDED NETWORK PROBLEMS

by

Fred Glover*

and

Darwin Klingman**

December 1977

(Revised January 1979)

Supersedes A055156 mc

* Professor of Management Science, University of Colorado, Boulder, CO 80309.

** Professor of Operations Research and Computer Sciences, BEB 608, The University of Texas at Austin, Austin, TX 78712.

This research was partially supported by FEA contract CR-03-70128-00 with Analysis, Research, and Computation, Inc., and by ONR Projects NR047-172 and NR047-021 with the Center for Cybernetic Studies, The University of Texas at Austin.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas at Austin
Austin, TX 78712
(512) 471-3322

ABSTRACT

This paper develops a special partitioning method for solving LP problems with embedded network structure. These problems include many of the large-scale LP problems of practical importance, particularly in the fields of energy, scheduling, and distribution. The special partitioning method, called the simplex special ordered network (SON) procedure, applies to LP problems that contain both non-network rows and non-network columns, with no restriction on the form of the rows and columns that do not exhibit a network structure. These LP/embedded network problems include as a special case multi-commodity network problems and constrained network problems previously treated in the literature, by simultaneously encompassing both side constraints and side activities.

The simplex SON procedure solves these problems by exploiting the network topology of the basis, whose properties are characterized by means of a specially constructed *master basis tree*. A set of fundamental exchange rules are developed which identify admissible ways to modify the master basis tree, and hence the composition of the partitioned basis inverse. The simplex SON method implements these rules by a set of streamlined labeling algorithms, while maintaining the network portion of the basis as large as possible, thereby accelerating the basis exchange step. As a result, LP/embedded network problems can be solved with less computer storage and significantly greater efficiency than available from standard linear programming methods.

Preliminary computational results are reported for an all-FORTRAN

implementation of the simplex/SON algorithm called PNET/LP. The test problems are real-world models of physical distribution and scheduling systems. PNET/LP has solved problems with 6200 rows and 22,000 columns in less than 6 minutes, counting all I/O, on an AMDAHL V-6 with a FORTRAN G compiler.

1.0 INTRODUCTION

The dramatic successes of the past several years in solving pure network problems [2, 3, 5, 6, 8, 19, 30, 41] have motivated consideration of methods for solving more general linear programming (LP) problems with embedded network structure. For example, in the realm of pure networks (capacitated minimum cost flow problems), the computational study [21] demonstrates that special purpose network computer codes are 150-200 times faster than the state-of-the-art LP code, APEX III. Subsequent studies of "singly constrained" networks (LP problems consisting of a network plus one additional side constraint) demonstrated that specialized methods also yield substantial computational advantages for problems that do not exhibit pure network structures, but which are "almost networks." The papers [18, 20, 31] show that these problems can be solved 25-50 times faster than APEX-III. Many practical LP problems, however, contain embedded networks with multiple side constraints and multiple side variables, and so it is extremely important to determine whether an efficient specialized method can be developed for these problems. The purpose of this paper is to describe such a method, called the simplex special ordered network (SON) algorithm. The simplex SON algorithm is a primal basis partitioning method that employs special updating and labeling procedures to accelerate computations involving the network-LP interface. A preliminary FORTRAN implementation of this method solves real-world physical distribution models 25-50 times faster than APEX-III, confirming that it is possible to create a marriage of network and LP methodology that has ad-

vantages for more general problems with embedded network structure.

For definitional purposes, we refer to an LP/embedded network problem as a capacitated or uncapacitated linear program in which the coefficient matrix A can be characterized as follows. The A matrix contains m + q rows and n + p columns which are ordered and scaled such that each column of the m x n submatrix A_{mn}, consisting of the first m rows and n columns of A, has at most one +1, one -1, and zeros elsewhere. A major portion of the LP literature has been devoted to problems in which: (a) m = n = 0 (standard LP problems); (b) p = 0 (multicommodity networks and constrained network problems); (c) p = 0 and the submatrix A_{mn} contains only one non-zero entry per column (LP/generalized upper bounding (GUB) problems); (d) p = q = 0 (pure network problems).

The success with special classes of LP/embedded network problems, already noted, has led to speculations [10, 18, 21] that good results can also be obtained by extending these ideas for problems where p and q are less than n and m, respectively. Motivation for such an extension that leads to a highly efficient implementation has come from a number of the major practitioners of linear programming including Harvey Greenberg, Milton Gutterman, Alan Goldman, and A. C. Williams. In addition, the members of SHARE, and a number of industrial and governmental agencies that have large-scale LP/embedded network problems, have strongly stressed the need for such a method. Several of these individuals and groups have urged us to undertake such a development, leading to the results reported in this paper. One of the important applications to which the simplex SON method is relevant is the national energy model PIES, developed by the

Federal Energy Agency [24]. In the PIES model, q = 0, m = 2500, n = 4400, and p = 4500.

In general, it is our experience that most large-scale LP problems involving production scheduling, physical distribution, facility location, personnel assignment, or personnel promotion contain a large embedded network component, sometimes consisting of several smaller embedded networks. Coupling constraints (q > 0) arise, for example, from economies of scale, limitation on the total number of promotions, capacity restrictions on modes of transportation (e.g., pipelines, barges), limitations on shared resources, multiple criteria, or from combining the outputs of subdivisions to meet overall demands. Coupling columns (p > 0) arise from activities which involve different time periods (e.g., storage), production alternatives (e.g., refinery activities), or which involve different subdivisions (e.g., assembly). For example, Agrico Chemical Fertilizer Company has physical distribution and facility location problems where p = 0, m = 6200, n = 22,000, and q = 20.

1.1 History of Methods for Solving Special Classes of LP/Embedded Network Problems

There are two basic approaches which have been employed to develop specialized techniques for the above special classes (cases b, c, and d) of LP/embedded network problems--*decomposition and partitioning methods*. Decomposition approaches are further characterized as *price-directive* or *resource-directive*. The papers [3, 11, 12, 13, 16, 43, 44, 45, 47] give variations of price-directive decomposition and the papers [3, 14, 34, 40, 42] are resource-directive decomposition.

Partitioning approaches can be divided into partitioning *block diagonal* structured linear problems or general partitioning to exploit *embedded substructure* within general linear problems. Because they share the same basic principles, we will briefly review the literature of both.

The general idea of partitioning block diagonal structured programs was originally proposed by Dantzig [14a] and in a slightly different setting, by Charnes and Cooper [9, vol. 2]. Later Bennett [7], Hartman and Lasdon [26], Heesterman [28], Kaul [33], and Weber and White [46] independently developed primal solution procedures of more general scope. The paper by Hartman and Lasdon [26] contains an excellent description of this approach and procedures for handling the working inverse. Further, their paper contains computational results of such an algorithm. Grigoriadis and Ritter proposed a dual method [25].

Dantzig and Van Slyke [15] then proposed their well known "GUB" specialization of the primal simplex algorithm for the case where each block contains only one row. The general block diagonal procedures were further refined and specialized for multicommodity network problems by Saigal [38]. This specialization involves carrying a working basis inverse whose size need not exceed the number of saturated arcs. Hartman and Lasdon [27] developed efficient procedures for updating this working basis inverse. However, neither Saigal nor Hartman and Lasdon discuss how this procedure may be efficiently implemented. Maier [36] refined their procedures and initiated implementation discussions. Kennington, et al [1, 29, 30] streamlined the implementation procedures of Maier and conducted extensive computational testing.

Charnes and Cooper [9, vol. 2] originally proposed partitioning to exploit embedded substructure within general linear problems in their Double Reverse Method. Bakes [4] independently proposed an analogous algorithm when p = 0. Klingman and Russell [35] developed additional specializations for exploiting pure network substructure and Hultz and Klingman [31, 32] for generalized network substructure (i.e., where each column of A_{mn} may have at most two arbitrary non-zero coefficients). These papers initiated the first in-depth discussion of implementation techniques. The paper by Glover, Karney, Klingman, and Russell [20] presents the first computational results of such an algorithm for pure network substructure and the paper by Hultz and Klingman [31] presents the first computational results for generalized network substructure. The work by McBride [37] and Graves and McBride [23] on Factorization subsequently redeveloped and refined the general procedures of [9, 14a]. Further, McBride's dissertation [37] discussed specialization of these procedures for exploiting pure network substructure.

1.2 Form of the Simplex SON Method for LP/Embedded Network Problems

The simplex SON method constitutes a highly efficient way to modify and implement the steps of the primal simplex algorithm for the completely general case of embedded pure network problems (where p > 0 and q > 0). The efficiency is the direct result of exploiting the pure network portion A_{mn} of the coefficient matrix and the network-LP interface by special labeling and updating procedures.

The starting point for the algorithm, following the natural design of partitioning methods, is to subdivide the coefficient matrix into network

and non-network components. By reference to this subdivision, a basis inverse compactification procedure is employed that maintains a *working basis inverse*, V^{-1}, whose dimension equals m + q less the rank of the basic subcolumns of A associated with A_{mn}. The size of V^{-1} therefore varies dynamically. This is one of the major features of this algorithm that distinguishes it from partitioning algorithms designed for constrained networks [31, 32, 35].

The basic variables not associated with V^{-1} are stored in a special graph form called the *master basis tree*. The development of the master basis tree and the procedures for using it to efficiently replace arithmetic operations constitute the principal contributions of this paper. We show that the operations normally performed by using the full basis inverse can instead be performed by special labeling and graph traversal techniques [5] applied to the master basis tree and its interface with V^{-1}. The organization of the simplex SON method maintains the network portion of the basis as large as possible at each iteration, thereby enabling these labeling and list processing procedures to operate on a maximally dimensioned part of the basis. This in turn minimizes the size of V^{-1}.

The resulting advantages over the standard LP implementation approach are several. First, the graph traversal operations reduce both the amount of work needed to perform the algorithmic steps and the amount of computer memory required to store essential data. Second, the algorithm orients the execution of operations in a manner that is best suited to the design of computers (making extensive use of linked list structures, pointers, and logical operaions in place of arithmetic operations.) Third, the method is less susceptible to round-off error and numerical inaccuracy. Most of

the operations are performed using original problem data and only the residual portion of the basis inverse associated with V^{-1} is subject to the slower customary updating, with its greater attendant susceptibility to round-off error and numerical inaccuracy. (The graph traversal procedures also automatically eliminate checking or performing arithmetic operations on zero elements.)

2.0 PROBLEM NOTATION

The LP/embedded network problem and its dual may be stated mathematically as follows:

Primal

Minimize $$c_n x_n + c_p x_p \tag{1}$$

subject to:

$$A_{mn} x_n + A_{mp} x_p = b_m \tag{2}$$

$$A_{qn} x_n + A_{qp} x_p = b_q \tag{3}$$

$$0 \leq x_n \leq u_n \tag{4}$$

$$0 \leq x_p \leq u_p \tag{5}$$

Dual

Maximize $$w_m b_m + w_q b_q - \gamma u_n - \Psi u_p \tag{6}$$

subject to

$$w_m A_{mn} + w_q A_{qn} - \gamma \leq c_n \tag{7}$$

$$w_m A_{mp} + w_q A_{qp} - \Psi \leq c_p \tag{8}$$

$$w_m, w_q \text{ --unrestricted} \tag{9}$$

$$\gamma, \Psi \geq 0 \tag{10}$$

where A_{mn} is (m x n), A_{mp} is (m x p), A_{qn} is (q x n), and A_{qp} is (q x p). The remaining vectors are conformable vectors whose subscripts indicate their dimensionality.

Each column of the matrix A_{mn} contains at most one +1, one -1, and zeros elsewhere. Thus A_{mn} corresponds to a pure network problem. For this reason the (2) portion of the LP/embedded network problem will be referred to as *node constraints* or simply *nodes*. The x_n variables will be referred to as *arc variables* or simply *arcs*. The arcs will be further classified as *ordinary arcs*, which have exactly two non-zero entries in A_{mn}, and as *slack arcs*, which have exactly one non-zero entry in A_{mn}.

In graph terminology, a *simple graph* G(*V*,E) is a finite set of vertices *V* and a finite set of edges E connecting the vertices. Each element of E is identified with an unordered pair of distinct elements of *V*. Schematically, each edge connects two distinct vertices, which are then considered to be adjacent. If the edge set E is expanded to contain edges which have both endpoints incident on the same vertex or multiple edges connecting the same two vertices (parallel edges), then G(V,E) is called a *general graph*. If each edge of the general graph has an implied direction then the graph is sometimes called a *digraph* or *directed graph*, the vertices are referred to as nodes, and the edges are referred to as arcs.

The underlying pure network problem A_{mn} defines one or more connected digraphs as follows. Each row of A_{mn} corresponds to a node and each column to an arc. The -1 entry in a column indicates the node where the arc begins (from node) and the +1 entry in a column indicates the node where the arc ends (to node). If a column has only one non-zero entry (a slack arc) the endpoints of the arc are incident on the same node. This representa-

tion of A_{mn} may consist of several disjoint connected digraphs. Because each arc is associated with a variable, it has lower and upper bounds and an objective function coefficient.

3.0 BASIS STRUCTURE

Using the standard bounded variable simplex algorithm, a basis B for the LP/embedded network problem is a matrix composed of a linearly independent set of column vectors selected from the coefficient matrix

$$A = \left[\begin{array}{c|c} A_{mn} & A_{mp} \\ \hline A_{qn} & A_{qp} \end{array}\right] .$$

The variables associated with the column vectors of B are considered to be basic variables x_B and all others are non-basic variables x_N at their lower or upper bound.

Without loss of generality, it will be assumed that A has full row rank. Any basis B for the LP/embedded network problem will, therefore, be a nonsingular matrix of order $(m + q) \times (m + q)$. Clearly, any basis matrix B may be partitioned as follows:

$$B = \begin{array}{c} \begin{array}{cc} x_{B1} & x_{B2} \end{array} \\ \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] \end{array} \qquad (11)$$

where B_{11} is a nonsingular submatrix of A_{mn}. Thus the basic variables x_{B1} associated with the $\begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix}$ columns are exclusively arc variables. The basic variables x_{B2} associated with the $\begin{bmatrix} B_{12} \\ B_{22} \end{bmatrix}$ columns may also contain arc variables. (x_{B1} and x_{B2} are written above their associated

components of B in (11).)

Based on the indicated partitioning of (11), the basis inverse B^{-1} may be written as follows:

$$B^{-1} = \left[\begin{array}{c|c} B_{11}^{-1} + B_{11}^{-1} B_{12} V^{-1} B_{21} B_{11}^{-1} & -B_{11}^{-1} B_{12} V^{-1} \\ \hline -V^{-1} B_{21} B_{11}^{-1} & V^{-1} \end{array}\right] \quad (12)$$

where $V = B_{22} - B_{21} B_{11}^{-1} B_{12}$.

The motivation for this way of partitioning (11) is to factor out the submatrix B_{11} of A_{mn} and thereby exploit its inherent triangularity by viewing and storing B_{11} as a digraph. This handling of B_{11} has several advantages: (1) the graph contains only the nonzero components of B_{11}, (2) any operations involving B_{11} may be performed by traversing the associated digraph using appropriately designed labeling techniques, (3) since B_{11} contains only original problem data, numerical errors are reduced.

The only matrices required to generate the basis inverse, as seen from (12), are B, B_{11}^{-1} and V^{-1}. The efficiency of generating the needed components of B^{-1} in performing the steps of the simplex method depends on the size and techniques used to store B_{11}, the labeling procedures used with B_{11} to eliminate matrix multiplications involving B_{11}^{-1}, and the procedures used to maintain the position of B.

The developments of the following sections show how to handle these considerations effectively.

3.1 Graph Representation of B_{11}

As in the case of A_{mn}, B_{11} defines a digraph. The structure of this digraph is a set of disjoint spanning trees, each of which is augmented by an arc whose from node and to node are the same node in B_{11}. Such an

arc of B_{11} will be called a *simple loop*. (This structure implies that B_{11} is a block diagonal matrix and each block is triangular. The blocks each consist of a spanning tree plus a simple loop. Thus, each block is a *quasi-tree*.) This structural property of B_{11} follows directly from the fact that B_{11} is a square nonsingular matrix and that B_{11} is a submatrix of A_{mn}. Thus each column of B_{11} contains at most two non-zero unit entries of the opposite sign.

A column of B_{11} corresponding to a simple loop may be of two types according to whether the associated column of A_{mn} is a slack arc or an ordinary arc. If the A_{mn} column is an ordinary arc, then the partitioning of B has split this arc between B_{11} and B_{21}--that is, one of its unit entries lies in B_{11} and the other in B_{21}. Because of this, the algorithm stores and uses B_{11} by keeping a larger digraph than the one corresponding to B_{11}. This larger digraph, called the *master basis tree*, contains every node in A_{mn} plus another node called the *master root*; thus, it always contains m + 1 nodes and m arcs. The nodes of this tree that correspond to rows of B_{21}, since they are external to the nonsingular network structure of B_{11}, are called *externalized roots* (ER's).

The master basis tree contains all of the ordinary arcs in B_{11}. Simple loops in B_{11} are contained in the master tree in a modified form. If the simple loop is a slack arc of A_{mn}, then the simple loop is replaced by an arc between the master root and its unique node. If the simple loop is an ordinary arc in A_{mn} it is replaced by an arc between its nodes in A_{mn}. Such arcs, thus, join ER's to nodes in B_{11}. To complete the master basis tree each ER is connected to the master root by an *externalized arc* (EA). Figure 1 graphically depicts a master basis tree.

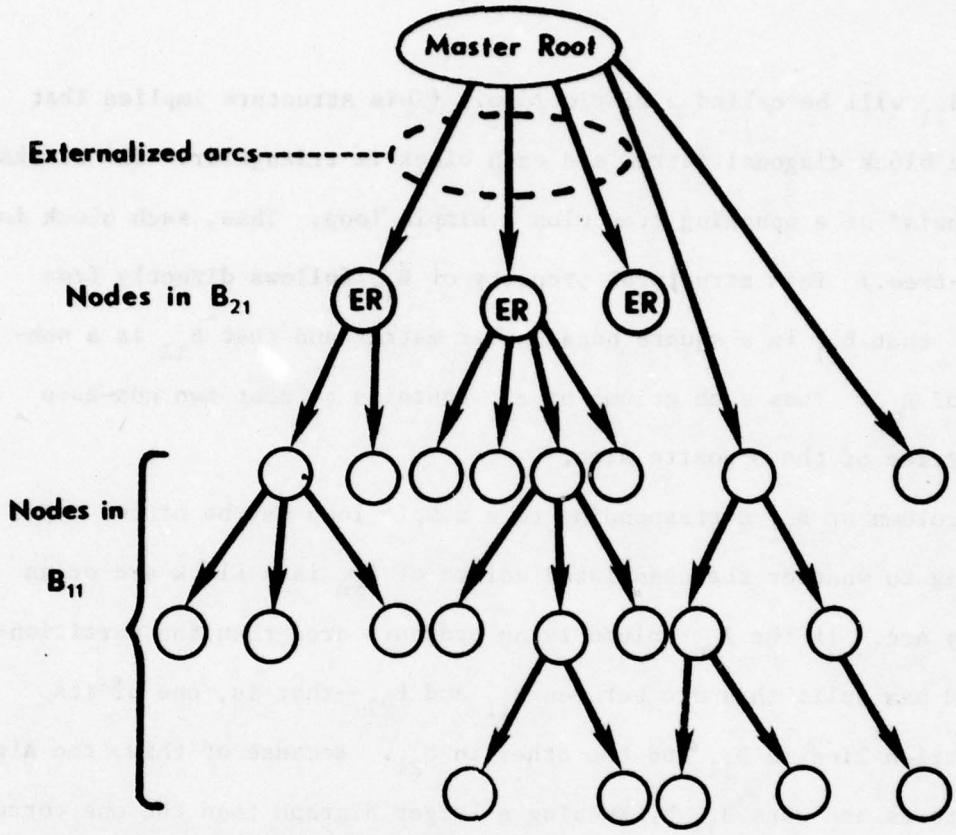


Figure 1 - Master Basis Tree

It is important to stress that the master basis tree is a conceptual scheme designed to allow the simplex/SON algorithm to efficiently maintain the partitioning of B while keeping the B_{11} portion at maximum size during each iteration. This construction should not be confused with the simple model device sometimes employed in pure network settings, where a pseudo root is added for the purpose of giving each slack arc two endpoints. The connections represented by the master basis tree include both network and "extra network" structures (mediated by externalized roots and arcs), and the rules for operating on these structures are of a very special type.

To understand the function of these rules, it should be noted that a multiplicity of cases must be considered when executing a basis exchange step. For example, if an arc variable is to be added to the basis, or transferred from x_{B2} to x_{B1}, the endpoint(s) of the arc may consist of (1) two nodes in a single block (quasi-tree) in B_{11}, (2) nodes in different blocks (quasi-trees) in B_{11}, (3) a node in B_{21} and a node in B_{11}, (4) two nodes in B_{21}, (5) a single node in B_{11}, (6) a single node in B_{21} (where cases (5) and (6) apply to slack arcs of A_{mn}). A similar multiplicity of cases applies to removing an arc from the basis, and the combinations that result from both adding and removing arcs are still more numerous.

The use of the master basis tree permits all of these cases to be unified in a particularly convenient fashion. The rules characterizing the conditions for adding and deleting arcs, and specifying the appropriate restructuring of the master basis tree, are as follows.

3.2 Fundamental Exchange Rules

1. An arc of x_{B2} can admissably be added to the B_{11} portion of the basis, without deleting another, if an only if its loop in the master basis tree contains at least one EA. (Such a loop can contain at most two EA's.) The updated form of the master basis tree then occurs in the following manner: (a) Add the new arc and drop any EA from the loop. (b) Change the status of the ER formerly met by the dropped EA to that of an ordinary node, transferring its row from the B_{21} to the B_{11} portion of the basis.

2. An arc can be deleted from B_{11} (removing a component of x_{B1}) without adding another as follows: (a) Identify the node of the selected arc that is farthest from the master root. (b) Change this node into an ER node by moving this node to B_{21} and attaching it to the master root by a newly created EA. At the same time delete the selected arc from B_{11}.

3. An arc can be added to B_{11} and another simultaneously removed from B_{11} as follows: (a) If the loop in the master basis tree created by the added arc includes the arc to be dropped, then the exchange step is handled exactly as an exchange step of an ordinary network basis. (Thus no EA's are added or dropped, and no nodes alter their status as ordinary nodes or ER nodes.) (b) If the loop in the master basis tree created by the added arc does not include the arc to be dropped, then the exchange may be performed as a two-part process that applies the preceding rules 1 and 2 in either order (as long as the exchange is valid).

4. B_{11} and B_{21} can be restructured, without adding or deleting basis arcs x_{B1}, by an exchange step that drops any EA and adds another EA to any node of the isolated tree (excluding the master root) created by dropping the first. This step is accomplished by interchanging the ER status and ordinary node status of two nodes which swaps their corresponding rows in B_{11} and B_{21}.

It should be remarked that the EA's have a special interpretation in these rules. Since the master basis tree spans all nodes of A_{mn}, and always contains the same total number of arcs (including EA's), the number of EA's corresponds to the number of non-arc variables in the basis (elements of x_{B2}) that are required to give the basis full row rank for the

A_{mn} portion of the problem.

It may also be observed that the A_{mn} portion of each column of B_{21} (the portion associated with the ER's) contains at most one non-zero entry. In particular, this partial column is the zero vector if its associated arc (element of x_{B1}) does not meet an ER, and otherwise contains a -1 entry if the from node of the arc meets an ER and a +1 entry if the to node of the arc meets an ER. No slack arcs of A_{mn} can contain entries in B_{21}, or else their B_{11} columns would be zero vectors.

However, it is possible for slack arcs of A_{mn} to have entries in B_{22}, and also for ordinary arcs of A_{mn} to have both of their non-zero entries in B_{22} (meeting two ER's at their endpoints). In this case such a slack arc or ordinary arc, when added to the master basis tree, creates a loop that includes an EA, and thus can be moved from x_{B2} to x_{B1} by the Fundamental Exchange Rules.

The validity of the Fundamental Exchange Rules is expressed in the following result.

Theorem 1: B_{11} is maintained as a nonsingular matrix by the addition and deletion of arcs if and only if the Fundamental Exchange Rules are applied.

Proof: The master basis tree maintains a linearly independent super-structure which, by rooting each block (quasi-tree) of B_{11} at an ER, and each slack arc of A_{mn} in B_{11} at the master root, assures that B_{11} is non-singular. Further, it is readily verified that each operation prescribed by the Fundamental Exchange Rules preserves these structural relationships. The primary requirement is to determine that none of the possible ways of modifying B_{11} is overlooked. We will not trace the details of a

full itemization of cases, since they are somewhat tedious, but simply remark that each of the alternatives can be directly observed to be handled correctly by the unifying construction of the master basis tree. QED.

It may be noted that Rule 4 of the Fundamental Exchange Rules, while not directly concerned with the addition and deletion of arcs, can affect the density of B_{21}, and indirectly the composition of V. The density of any row of B_{21} associated with A_{mn} can be minimized (reduced to a single non-zero, if any non-zeros exist) by a single application of Rule 4 to swap the status of the associated ER node with that of the last node of its subtree (i.e., swapping a row of B_{11} with a row of B_{12}), using the last node function [5]. Note that by successively applying this procedure each ER could be structured so that it has only one arc of A_{mn} incident on it.

More important and compelling is the issue of whether there exists an easily specifiable way to apply the Fundamental Exchange Rules to maximize the dimension of B_{11} (the number of components of x_{B1}) or whether, due to the influence of the non-network basis structure, a particular configuration for B_{11} cannot be augmented except by a complex strategy of removing current arcs and adding new ones. The resolution of this issue, which relies on the fact that maximizing the dimension of B_{11} is equivalent to minimizing the number of EA's in the master basis tree, is expressed in the following result.

Theorem 2: The dimension of B_{11} is maximized (and the number of EA's minimized) by successively applying Rule 1 of the Fundamental Exchange

Rules until no more arcs are admissible to be added by this rule.

Proof: Suppose instead a master basis tree is obtained that cannot admit the addition of any arc by Rule 1, yet that another master basis tree exists with fewer EA's. Since any tree can be transformed into any other by some sequence of pairwise arc exchanges, producing a tree at each stage, we can find some tree T_1 in such a sequence which allows no swap reducing the number of EA's, and another tree T_2, obtained by a swap in T_1, such that a swap in T_2 will reduce the number of EA's. (Any single swap that reduces the number of EA's is an application of Rule 1.) Thus, the tree with a smaller number of EA's is two swaps away from T_1. But a basic result of tree exchanges is that if two trees are two swaps apart, then the swaps can be executed in either sequence, producing a tree at each step (see, e.g., [22]). This proves the theorem by contradiction. QED.

Theorem 2, which applies either to adding incoming arcs to x_{B1}, or transferring arcs from x_{B2} to x_{B1}, also makes it possible to maintain B_{11} at its maximum dimension at each iteration of the primal simplex method, as will be demonstrated subsequently. For the special case in which the LP/embedded network problem has no non-network constraints (though any number of non-network variables), the following observation is useful.

Corollary 1: If the nodes of A_{mn} span all the rows of A (i.e., if A_{qn} is empty), then every basic arc variable can be included in x_{B1}.

Proof: Any basis arc variable that cannot be included in x_{B1} must create a cycle that does not include an EA. This implies linear dependence in the arc variable columns of B. These columns must contain zeros in B_{12}

and B_{22}; otherwise their arcs would intersect an ER and their loop would contain an EA. Then B must be singular, contrary to the fact that it is a basis.

The value of this Corollary is that it allows all basic network arcs automatically to be included in x_{B1} for LP/embedded network problems without side constraints ($q = 0$), avoiding the work of checking to see whether the inclusion of any particular arc is admissible.

3.3 Labeling Algorithms for Accelerating Basis Computations

Drawing on the network topology of B, as embodied in the construction of the master basis tree, we turn now to the determination of special algorithms for processing this master tree. The algorithms are specifically designed to carry out computations involving B^{-1} that are required in the steps of the primal simplex method (pricing out the basis, determining the representation of the incoming variable, etc.). In particular, we are concerned with identifying the most effective way to take advantage of the network structure of B_{11} embedded in the partitioned inverse.

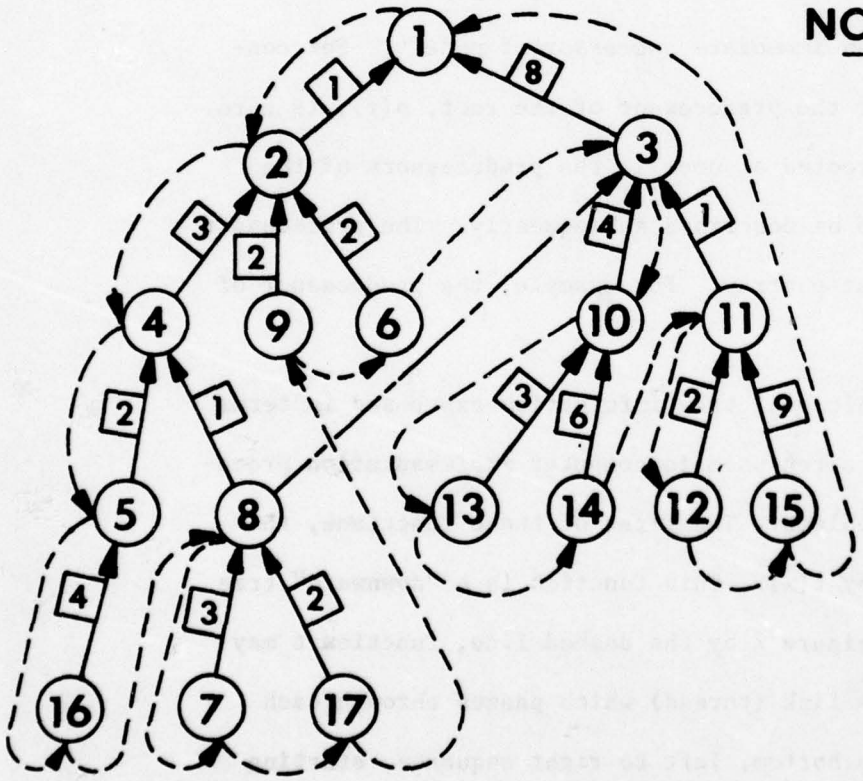
The principal computations of the primal simplex method that involve B_{11}^{-1} can be segregated into three classes concerned with computing: (1) $B_{11}^{-1}G$, (2) HB_{11}^{-1}, and (3) $HB_{11}^{-1}G$. By allowing H and G in each of these classes to be either a vector or a matrix (and in the case of a vector, to have either single or multiple non-zero components), it is possible to express the generic forms of all of the simplex method calculations involving B_{11} in the partitioned inverse. In this section we provide the procedures capable of performing these calculations in a manner that most fully exploits the structure of B_{11}. Significantly, it

turns out that the most effective list processing and labeling procedures differ in each of the several alternative cases. The list structures and functions used in these procedures are those commonly employed in network optimization. For completeness, we include explicit descriptions of these lists.

To provide a common visual frame of reference, the root node r may be viewed as the highest node in the tree with all the other nodes hanging below it. The tree is then represented by keeping a pointer list which contains for each node w (other than the root) the unique node v above node w which is connected to w by an arc. This upward pointer is called the *predecessor* of node w and will be denoted by p(w). Correspondingly, node w is called an *immediate successor* of node v. For convenience, we will assume that the predecessor of the root, p(r), is zero. Figure 2 illustrates a tree rooted at node 1, the predecessors of the nodes, and other functions to be described subsequently. The predecessor of a node is identified in the p array. For example, the predecessor of node 16 is node 5.

Figure 2 illustrates additional tree information expressed in terms of node functions, which are often used in computer implementation procedures for solving network problems. The first of these functions, the *thread* function, is denoted by t(x). This function is a "downward" tree pointer. As illustrated in Figure 2 by the dashed line, function t may be thought of as a connecting link (thread) which passes through each node exactly once in a top to bottom, left to right sequence, starting from the root node. For example, in Figure 2, t(1) = 2, t(2) = 4, t(4) = 5, t(5) = 16, t(16) = 8, etc.

Predecessor	**p(x)**
Node potential	**d(x)**
Thread	**t(x)**
Reverse thread	**rt(x)**
Depth	**dh(x)**
Cardinality	**c(x)**
Last node in subtree	**f(x)**



NODE	p	d	t	rt	dh	c	f
1	0	0	2	15	0	17	15
2	1	1	4	1	1	9	6
3	1	8	10	6	1	7	15
4	2	4	5	2	2	6	17
5	4	6	16	4	3	2	16
6	2	3	3	9	2	1	6
7	8	8	17	8	4	1	7
8	4	5	7	16	3	3	17
9	2	3	6	17	2	1	9
10	3	12	13	3	2	3	14
11	3	9	12	14	2	3	15
12	11	11	15	11	3	1	12
13	10	15	14	10	3	1	13
14	10	18	11	13	3	1	14
15	11	18	1	12	3	1	15
16	5	10	8	5	4	1	16
17	8	7	9	7	4	1	17

FIG. 2 - TREE LABELING TECHNIQUES

Letting n denote the number of nodes in the tree, the function t satisfies the following inductive characteristics:

a) The set $\{r,t(r),t^2(r),\ldots,t^{n-1}(r)\}$ is precisely the set of nodes of the rooted tree, where by convention $t^2(r) = t(t(r))$, $t^3 = t(t^2(r))$, etc. The nodes $r,t(r),\ldots,t^{k-1}(r)$ will be called the *antecedents* of node $t^k(r)$.

b) For each node i other than node $t^{n-1}(r)$, t(i) is one of the nodes such that p(t(i)) = i, if such nodes exist. Otherwise, let x denote the first node in the predecessor path of i to the root which has an immediate successor y and y is not an antecedent of node i. In this case, t(i) = y.

c) $t^n(r) = r$; that is, the "last node" of the tree threads back to the root node.

The *reverse thread* function, rt(x), is simply a pointer which points in the reverse order of the thread. That is, if t(x) = y, then rt(y) = x. Figure 2 also lists the reverse thread function values.

The *depth* function, dh(x), indicates the number of nodes in the predecessor path of node x to the root, not counting the root node itself. If one conceives of the nodes in the tree as arranged in levels where the root is at level zero, and where all nodes "one node away" from the root are at level one, etc., then the depth function simply indicates the level of a node in the tree. (See Figure 2.)

The *cardinality* function, c(x), specifies the number of nodes contained in the subtree associated with node x. By the nodes in the subtree associated with node x, we mean the set of all nodes w such that the predecessor path from w to the root contains x. (See Figure 2.)

The *last subtree node* function, f(x), specifies the node in the subtree of x that is encountered last when traversing the nodes of this subtree in "thread order." More precisely, f(x) = y where y is the unique node in the subtree of x such that t(y) is not also a node in the subtree of x. (See Figure 2.)

Note that both the domain and the range of each of the above discrete functions consist of a subset of the problem nodes and thus are independent of the number of arcs in the LP/embedded network problem. Since the master basis tree contains m + 1 nodes, a one-dimensional array of size m + 1, called a *node length array*, is allocated to each function during computer implementation. The procedures for updating the values of the functions when the tree is reconfigured are discussed in [5, 16].

In computing a particular vector $x = B_{11}^{-1}G$, the components of x are associated with columns of B_{11} via the equation $B_{11}x = G$, and are thus associated with arcs. Similarly, the components of the vector $W = HB_{11}^{-1}$ are associated with rows of B_{11} via the equation $WB_{11} = H$ and are thus associated with nodes. Consequently, in computing these vectors we will let:

x_k = the component of x associated with the basis arc k (whose endpoints are nodes k and p(k))

W_k = the component of W associated with node k.

These latter definitions will be modified only to allow x and W to represent matrices rather than vectors, in which case each row of x will be thought of as a row of variables associated with the corresponding arc, and each column of W as a column of variables associated with the corresponding node.

Finally, we will refer to a basis arc as *conformable* if its A_{mn} direction agrees with its basis predecessor orientation (i.e., the from node of the arc lies at the predecessor node and the to node of the arc lies at the successor node), and refer to the arc as *nonconformable* otherwise.

A. Algorithms for computing $x = B_{11}^{-1}G$ (solving for x in the equation $B_{11}x = G$).

A1. G is a column vector with one non-zero element. Let G_o equal the non-zero element G_k of G, associated with node k.

A1.1 Let q = p(k). Let $x_k = G_o$ if the arc is conformable, and let $x_k = -G_o$ if the arc is nonconformable.

A1.2 If q is a root (either an ER or the master root) stop. All non-zero elements of x have been assigned the proper value. Otherwise, let k = q and return to A1.1.

A2. G is a column vector with two non-zero elements. Use the depth or cardinality function in conjunction with the predecessor so that the trace from the two nodes that correspond to the non-zeros of G can be interrupted at their intersecting paths.

A2.1 Apply A1 to each non-zero, independently. Stop by the criterion of A1.2 if the paths do not intersect first.

A2.2 If the paths intersect before either meets a root, temporarily stop at step A1.2 when q is the intersection node, and redefine G_o to be the sum of the two non-zeros of G. If G_o is zero, stop. Otherwise, continue A1 to its termination.

A3. G is a column vector with more than two non-zero elements.

Option 1. Apply a generalized form of A2, not proceeding beyond any path intersection until all paths meeting at that point have been traced to it. The value of G_o at that point becomes the sum of the non-zeros on the starting nodes of the paths meeting there. (This can be useful if most of the paths are known to intersect only at roots.)

Option 2. (Generally preferred.) Let G_i denote the i^{th} element of G (whether non-zero or not). Using the last node function, identify the last node of the master tree, and designate this node to be node k.

A3.1 If p(k) is not a root, execute Step A1.1 for G_o replaced by G_k, and let $G_q = G_q + G_k$. (A3.1 can be skipped if $G_k = 0$ thus calculating only non-zero x_k values.)

A3.2 Let k = rt(k). If k is the master root, stop. Otherwise, return to A3.1. (Note in this procedure, one may avoid checking whether p(k) is a root in A3.1 by allowing "fictitious" variables x_k to be associated with EA's.)

A4. G is a matrix (and hence x is also a matrix). Let G_i denote the i^{th} row of G and let x_i denote the i^{th} row of x. Then apply algorithm A3, Option 2.

B. Algorithms for computing $W = HB_{11}^{-1}$ (solving for W in the equation $WB_{11} = H$.)

B1. H is a row vector with one non-zero element. Let H_k denote the non-

zero element of H, which occurs in the k^{th} position, associated with the arc k of the basis tree joining nodes k and p(k).

B1.1 Let $H_o = H_k$ if the arc is conformable and let $H_o = -H_k$ otherwise. Let i = k.

B1.2 Let $W_i = H_o$.

B1.3 If, by the last node function, node i is the last node of the subtree headed by node k, stop. All non-zeros of W have been generated (with the value H_o). Otherwise, let i = t(i) and return to B1.2.

B2. H is a row vector with more than one non-zero element. If the subtrees containing the arcs associated with the non-zeros of H are known to be disjoint, apply B1 independently to each subtree. Otherwise, select any node k of the master tree (possibly an ER or the master root) which heads a subtree whose arcs "contain" all non-zeros of H. Definitionally, for the following, let $W_i = 0$ if node i represents a root node (the master root or an ER), and let $H_i = 0$ if arc i represents an externalized arc (which may always be regarded as conformable).

Option 1. Let $W_k = 0$ and i = t(k). Let k* denote the last node of the subtree headed by k.

B2.1 Let q = p(i). Then let

$$W_i = W_q + H_i \quad \text{if arc i is conformable}$$

$$W_i = W_q - H_i \quad \text{if arc i is nonconformable.}$$

B2.2 Let i = t(i). If i = k*, stop. All non-zero elements of W have been determined. Otherwise, return to B2.1.

Option 2. (Generally preferable if H has few non-zeros.) Define two lists, a subtree header list SH(j) and a last node list LN(j), j = 1,...,J. To begin, let SH(1) = k and let LN(1) = k*, the last node in the subtree headed by k. Let $W_k = 0$ and i = t(k).

B2.3 Let $H_o = H_i$ if arc i is conformable and let $H_o = -H_i$ otherwise.

B2.4 Let $W_i = H_o$. If i = k*, go to B2.7.

B2.5 Let i = t(i). If $H_i = 0$, go to B2.4. Otherwise, let $H_o = H_o + H_i$ if arc i is conformable and let $H_o = H_o - H_i$ if arc i is nonconformable.

B2.6 If i = k*, let $W_i = H_o$ and go to B2.7. Otherwise, let SH(J) = i if the last node in the subtree headed by i is k* (updating the header list to name the most recent node whose last node is k*). Otherwise, let J = J + 1, let SH(J) = i, let k* denote the last node in the subtree headed by i, and go to B2.4.

B2.7 If J = 1, stop. All non-zero components of W have been determined. Otherwise, let J = J - 1, let k (SH(J), let k* = LN(J) and let $H_o = W_k$. Then go to B2.5.

B3. H is a matrix (hence W is also a matrix). Let H_i denote the i^{th} column of H and let W_i denote the i^{th} column of W. Then apply algorithm B2, Option 1 or Option 2. (If many of the columns of H are 0 vectors, Option 2 is preferable. If, further, non-zero columns of H have few non-

zero elements, then a variant of Option 2 can be applied, particularly by using pointers that name only non-zero elements.)

Algorithms A1, A2, B1, and B2 Option 1 are direct counterparts of algorithms already used in network optimization methods. Algorithms A3, A4, B2 Option 2, and B3 are new, designed to handle the special requirements of the LP/embedded network problem, with the same types of computational advantages that have derived from their simpler predecessors. One of the principal features shared by all of these algorithms, which provides a primary basis for exploitation by the method of the next section, is given in the following result.

Theorem 3: The value assigned to a variable at any iteration of any of the preceding algorithms is the correct solution value for the indicated equation and is not modified at any subsequent iteration.

Proof: The fact that a solution value, once assigned, is not changed thereafter, can be ascertained by tracing the steps of the algorithms. That this value is correct, in the case of the new algorithms, follows from the structural characteristics of the master basis tree already established, and from the list processes employed in these algorithms (see e.g., [5]).

The usefulness of the "once and for all" determination of values indicated in Theorem 3 manifests itself in the simplex SON optimization procedure by providing the basis for three additional algorithms which, together with the algorithms preceding, yield the ability to accelerate the calculation of more complex matrix products. The basis of these algorithms lies in identifying the appropriate manner to accumulate a matrix

product at intermediate stages, conserving time and memory. The form of these algorithms is as follows.

C. Algorithms for computing $A = HB_{11}^{-1} G + Z_o$

C1. G is a column vector and H is a matrix. Begin with Z equal to the column vector Z_o.

C1.1 Compute $x = B_{11}^{-1} G$ by the appropriate member of the A algorithms (depending on the structure of G).

C1.2 As each non-zero element x_i of the column vector X is computed, let $Z = Z + x_i H_i$, where H_i is the i^{th} column of H.

C2. H is a row vector and G is a matrix. Begin with Z equal to the row vector Z_o.

C2.1 Compute $W = HB_{11}^{-1}$ by the appropriate member of the B algorithms.

C2.2 As each non-zero element W_i of the row vector W is computed, let $A = A + W_i G_i$, where G_i is the i^{th} row of G.

C3. G and H are both matrices. Begin with Z equal to the matrix Z_o.

Option 1.

C3.1 Compute $x = B_{11}^{-1} G$ by algorithm A4.

C3.2 As each non-zero row x_i of x is computed, let

$Z_k = Z_k + H_{ki} x_i$ for each row Z_k of Z where H_{ki} is the k^{th} element of the i^{th} column H_i of H.

Or alternatively,

$Z_j = Z_j + H_i x_{ij}$ for each column Z_j of Z where x_{ij} is the j^{th} component of x_i.

Option 2.

C3.3 Compute $W = HB_{11}^{-1}$ by algorithm B3.

C3.4 As each non-zero column W_i of W is computed, let

$Z_k = Z_k + W_{ki}G_i$ for each row Z_k of Z where W_{ki} is the kth element of W_i and G_i is the i^{th} row of G.

Or alternatively,

$Z_j = Z_j + W_iG_{ij}$ for each column Z_j of Z where G_{ij} is the j^{th} component of G_i.

The validity of these algorithms follows directly from the validity of the A algorithms and B algorithms, and from the admissible options for organizing matrix computations. A special consequence of these methods, which is uniform throughout all calculations of x, W, or Z, is as follows.

Corollary 2. The most efficient versions of the A, B, and C algorithms, when either G or H is a matrix, result by storing G by row and storing H by column.

Proof: Immediate from the structure of the algorithms.

This outcome has noteworthy implications for the unified implementation of the A, B, and C algorithms in the simplex SON precedure. In particular, the identity of G in the simplex SON procedure (when G is a matrix), is characteristically B_{12}, and the identity of H (when H is a matrix), is characteristically B_{21} or the augmented matrix $\begin{pmatrix} B_{21} \\ C_{B1} \end{pmatrix}$. Thus, by Corollary 2, it is preferable to store B_{12} by row and B_{21} by column. This constitutes a departure from the methodology of previous compact basis procedures. That is, the result that the most effective computa-

tion arises by storing different components of B differently, provides a new organizational strategy for compact basis methods.

With these foundations, we are now ready to specify the complete form of the simplex SON optimization method.

4.0 ALGORITHMIC STEPS

This section describes in detail the basic simplex operations as they pertain to the LP/embedded network problem. These operations include initialization, checking for optimality, finding the representation of the vector entering the basis, the basis exchange step, and calculating updated dual variable values. As noted in Section 3, it will be assumed that the basis is partitioned as in (11), that B_{11} is stored using a master basis tree, and that V^{-1} is the only portion of the basis inverse that is being kept.

4.1 Initialization

An initial basis B for the LP/embedded network problem can be obtained by selecting a set of variables whose columns in A_{mn} are linearly independent and then augmenting these columns by appending slack or artificial variables to the problem that result in satisfying the constraints (2) and (3). Columns for artificial variables whose unit entries occur in the first m rows can be treated as an augmentation of the A_{mn} matrix.

Given an initial basis, Fundamental Exchange Rule 1 can be used to partition the basis so that the dimensionality of B_{11} is maximum.

Next, $V = B_{22} - B_{21} B_{11}^{-1} B_{12}$ is computed by algorithm C3, letting

$Z_0 = B_{22}$, $H = -B_{21}$, and $G = B_{12}$. V^{-1} is, then, calculated.

Once a basis has been selected and partitioned, the complementary slackness conditions can be used to obtain dual variable values which satisfy:

$$(w_m, w_q) \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = (c_{B1}, c_{B2}) \tag{13}$$

where c_{B1} and c_{B2} are the vectors of objective function coefficients associated with x_{B1} and x_{B2}. In expanded form, (13) becomes:

$$w_m B_{11} + w_q B_{21} = c_{B1} \tag{14}$$

$$w_m B_{12} + w_q B_{22} = c_{B2} \tag{15}$$

Equations (14) and (15) may be rewritten as follows:

$$w_m B_{11} = c_{B1} - w_q B_{21} \tag{16}$$

$$w_q (B_{22} - B_{21} B_{11}^{-1} B_{21}) = c_{B2} - c_{B1} B_{11}^{-1} B_{12} \tag{17}$$

Noting that $V = B_{22} - B_{21} B_{11}^{-1} B_{21}$, (17) can be stated as:

$$w_q = (c_{B2} - c_{B1} B_{11}^{-1} B_{12}) V^{-1} \tag{18}$$

w_q could be recomputed at future iterations using (18) and algorithm C2 to find $-c_{B1} B_{11}^{-1} B_{12}$. However, as q (the number of constraints in (3)) increases, the number of operations involved in this process becomes prohibitive. Thus, for large q, it is better to treat w_q as an extra row of V^{-1} and update it at each iteration.

Given an initial value of w_q, (16) provides an excellent framework for the calculation of w_m. Once the right hand side of (16) has been calculated, w_m may be computed by algorithm B2.

4.2 Determination of Optimality

Let $P_j = \begin{bmatrix} P_{m,j} \\ P_{q,j} \end{bmatrix}$ denote the j^{th} column vector of the coefficient matrix A, where $P_{m,j}$ is associated with constraints (2), and $P_{q,j}$ is associated with constraints (3). Optimality of both the primal and dual solution occurs when the dual solution (as well as the primal solution) is feasible. A determination of this can be made by first calculating the updated objective coefficients c_j^* associated with each primal column vector as follows:

$$c_j^* = c_j - (w_m P_{m,j} + w_q P_{q,j}) \tag{19}$$

Dual feasibility is achieved when the following conditions are satisfied for all j:

$$\begin{aligned} &c_j^* \geq 0, \; x_j = 0 \\ &c_j^* = 0, \; x_j \text{ - basic} \\ &c_j^* \leq 0, \; x_j = u_j. \end{aligned} \tag{20}$$

If any one of the conditions in (20) is not satisfied, then the associated primal column vector may be selected to enter the basis.

4.3 Finding the Representation of the Entering Vector

Let $P_s = \begin{bmatrix} P_{m,s} \\ P_{q,s} \end{bmatrix}$ denote the column vector selected to enter the basis matrix. The representation $\alpha = \begin{bmatrix} \alpha_{B1} \\ \alpha_{B2} \end{bmatrix}$ of P_s in terms of B must be computed so that the vector to leave the basis can be determined. This computation involves solving the following system of equations:

$$\alpha_{B1} = B_{11}^{-1} P_{m,s} + B_{11}^{-1} B_{12} V^{-1} B_{21} B_{11}^{-1} P_{m,s} - B_{11}^{-1} B_{12} V^{-1} P_{q,s} \tag{21}$$

$$\alpha_{B2} = V^{-1}(-B_{21} B_{11}^{-1} P_{m,s} + P_{q,s}). \tag{22}$$

Substituting α_{B2} into (21) yields

$$\alpha_{B1} = B_{11}^{-1} (P_{m,s} - B_{12}\alpha_{B2}). \qquad (23)$$

It is thus efficient to first calculate α_{B2} and then to use this result to find α_{B1}. To compute α_{B2}, first compute the quantity within the parentheses of (22) by algorithm C1 and then multiply by V^{-1}. To compute α_{B1}, use algorithm A3.

Using this method, a ratio test should be performed immediately on α_{B2} before the computation of α_{B1} is made since degenerate pivots occur frequently in network problems and, therefore, unnecessary computations could be avoided.

4.4 The Basis Exchange Step

Let $P_r = \begin{bmatrix} P_{m,r} \\ P_{q,r} \end{bmatrix}$ denote the vector selected to leave the basis. This column is identified by the minimum ratio calculation. The updated form of this column, $B^{-1} P_r$, is a unit vector, whose non-zero entry occurs in the pivot row. To execute the basis exchange step, it is necessary to identify the segment of this row in B^{-1} that affects the updating of V^{-1}. There are two cases:

1. The outgoing variable is an element of x_{B2}. The relevant segment of the pivot row is simply the row of V^{-1} that corresponds to the outgoing variable.

2. The outgoing variable is an element of x_{B1}. In this case, the pivot row segment that affects the updating of V^{-1} does not lie in V^{-1} itself, but is contained in the upper right quadrant of the partitioned B^{-1}. Consequently, from (12), this row segment has the form

$$-e_i \; B_{11}^{-1} \; B_{12} \; V^{-1} \qquad (24)$$

where e_i is the unit row vector with a 1 in the position corresponding to the row of B^{-1} associated with the outgoing variable. The computation of (24) may be accomplished by first computing $-e_i \; B_{11}^{-1} \; B_{12}$, using algorithm C2 (and algorithm B1 in the initial step). The result is then multiplied by V^{-1}.

Once the appropriate pivot row segment is thus determined, the updating of V^{-1} (and of w_q) proceeds in the customary manner by a pivot step restricted to this portion of B^{-1}.

The final type of operation of the simplex SON method is the modification of V^{-1} by the addition or deletion of a row or column.

4.5 Changing the Dimensionality of V^{-1}

The dimensionality of V^{-1} is determined by the dimensionality of x_{B2} (hence also of x_{B1}), and this in turn rests on the application of the Fundamental Exchange Rules. As previously noted, the initial composition of x_{B1} and x_{B2} can be determined by successive application of Rule 1 to maximize the dimension of B_{11}, and consequently to minimize the dimension of V^{-1}. Once the simplex SON method begins with this condition satisfied, Theorems 1 and 2 imply that the basis exchange step can maintain this condition at each iteration by the transfer of at most one element between x_{B1} and x_{B2} (depending on the configuration of the master basis tree, which may be modified by the addition of an arc corresponding to the incoming variable or by the deletion of an arc corresponding to the outgoing variable).

In particular, by applying Rule 1, 2, or 3 of the Fundamental Exchange Rules to the incoming and outgoing variables (if one or both of them correspond to arcs), and by applying Rule 1 to a potential transfer variable (an element of x_{B2} corresponding to an arc) --or, if appropriate, applying Rule 3 to the transfer variable and the outgoing variable--the resulting reconfiguration of the master basis tree will automatically maximize the dimension of B_{11} and minimize the dimension of V^{-1}. In each case, V^{-1} will be modified by the addition or deletion of at most one row and at most one column (in each combination that leaves V^{-1} square). The objective, then, is to show how this modification of V^{-1} can be brought about.

The deletion of a row or column of V^{-1} may be accomplished in a straightforward manner. Therefore we address the operations involved in the addition of such a vector. (In each case, the additions should take place before the execution of the pivot steps, allowing the pivot step to determine their newly updated forms.)

Adding a row to V^{-1}. The only row that may be added to V^{-1} is that of the pivot row, and then only if it does not already lie in V^{-1} (i.e., only if the outgoing variable is an element of x_{B1}). The way to identify and calculate this updated row is by (24). Note that the addition of a row to V^{-1} does not entail any additional computation, since in this case the form of the pivot row must be determined in any event.

Adding a column to V^{-1}. Any column of B^{-1} not already overlapping V^{-1} must lie in the left half of the partitioned basis inverse (12). Consequently, the updated and original form of this column is given,

respectively, by $\theta = \begin{bmatrix} \theta_{B1} \\ \theta_{B2} \end{bmatrix}$ and $e_j = \begin{bmatrix} e_k \\ 0 \end{bmatrix}$, where the unit element of e_k and e_j occur in the same position. With this identification, we may compute θ from the explicit representation of the partitioned basis inverse (12) in precisely the manner used earlier to compute the basis representation α of the column for the incoming variable. In particular, the form of θ is obtained by substituting θ_{B1}, θ_{B2}, e_k and 0, respectively, for α_{B1}, α_{B2}, $P_{m,s}$, and $P_{q,s}$ in (22) and (23). This yields

$$\theta_{B1} = B_{11}^{-1} (e_k - B_{12} \theta_{B2}) \tag{25}$$

and

$$\theta_{B2} = V^{-1} B_{21} B_{11}^{-1} e_k \tag{26}$$

The calculation of θ_{B2} proceeds by first applying algorithm C1 to compute $B_{21} B_{11}^{-1} e_k$ (and using algorithm A1 in the initial step), followed by multiplication by V^{-1}. Then, unless both a row and a column are simultaneously to be added to V^{-1}, it is unnecessary to compute θ_{B1} since θ_{B2} is the portion of B^{-1} required.

On the other hand, if a row is to be added to V^{-1} as well as a column, it is necessary to compute the pivot row element in θ_{Bi}. Thus, letting e_i denote the unit row vector as previously defined in the generation of the updated pivot row, this element of θ_{B1} is given by

$$e_i B_{11}^{-1} (e_k - B_{12} \theta_{B2}) \tag{27}$$

This can be computed by algorithm C2, using algorithm B1 in the first part, and noting that the matrix stipulation for G may be replaced by the stipulation that G is a column vector.

The use of the A, B, and C algorithms in these calculations materially accelerates the steps of the simplex SON method, following the indicated prescriptions.

5.0 IMPLEMENTATION AND COMPUTATIONAL TESTING

5.1 Implementation

We have implemented a preliminary FORTRAN version of the simplex SON method for capacitated LP problems where $A_{mp} = 0$. This in-core code, called PNET/LP, employs super-sparsity (i.e., it stores only the unique non-zero elements of A) and keeps V^{-1} in product form. PNET/LP also employs the predecessor, thread, reverse thread, cardinality, and last node functions.

PNET/LP has been run on a CDC 6600, a CYBER-74, and an AMDAHL V-6. The computer memory space utilized by PNET/LP depends on several factors including (i) the computer, (ii) the number of unique non-zeros in the original problem, (iii) the number of unique non-zeros in the ETA file. Consequently, it is impossible to specify in simple terms an exact formula for the amount of memory required by the program. An approximate formula for IBM 370 and AMDAHL computers is 46 bytes per network row, 40 bytes per LP row, 12 bytes for each arc variable (i.e., for each element of x_n) plus 4 bytes for each non-zero coefficient in its LP rows, and 8 bytes for each non-arc variable (i.e., for each element of x_p) plus 4 bytes for each non-zero coefficient in its LP rows. In addition, PNET/LP keeps a variably dimensioned working space array which contains the pool of unique non-zero elements of A and V^{-1}.

PNET/LP first optimally solves the network portion of the LP/embedded network problem. This optimal network basis is then augmented by appropriate slack or artificial variables to form a starting basis for the entire problem. During the solution of the network problem, PNET/LP employs the modified row minimum start [19], and the standard Phase I-II method for handling artificial variables. If the optimal network basis is augmented by artificial variables, PNET/LP minimizes the sum of infeasibility in Phase I for the full LP/embedded network problems.

5.2 Computational Testing

In order to evaluate the computational merits of PNET/LP, we tested the following three classes of problems: (i) pure network, (ii) GUB/LP problems, and (iii) embedded network/LP problems where $A_{mp} = 0$ and m is large relative to q.

The first class, pure network, was selected in order to determine the relative efficiency of PNET/LP to a state-of-the-art special purpose code for solving network problems. To conduct our comparison we used the network code PNET-I of [19] and modified its pivot criterion to correspond to that of PNET/LP. Our analysis disclosed that PNET-I is only twice as fast as PNET/LP on pure network problems. This is surprising in view of the fact that PNET-I is 150-200 times faster than commercial LP codes such as APEX-III and MPSX-370. However, since PNET/LP is designed to exploit network structures, it is relevant to identify the reasons for the difference in speed between PNET/LP and PNET-I. These reasons are as follows:

1. PNET/LP uses double precision flowing point arithmetic while PNET-I uses integer arithmetic.

2. PNET/LP, due to its use of super-sparsity and its ability to solve any general LP problem, stores the original problem data in a more complex manner than PNET-I. Consequently, PNET/LP requires more time to access the original problem during basis exchanges and pricing operations.

3. For numerical reasons, PNET/LP uses the standard Phase I-II method, while PNET-I uses the BIG-M method. Computational testing [19] has shown that the BIG-M method is substantially more efficient for pure network problems. Because of these factors, the difference in solution speed between PNET/LP and PNET-I is much less than might be expected. In general, we found that PNET/LP is able to solve network problems with 2000 nodes and 9000 arcs in less than 15 seconds using a FORTRAN G compiler on an AMDAHL V6.

The second class of problems, GUB/LP problems, was selected because generalized upper bound constraints can be viewed as a very simple form of network constraints. Further, since the GUB feature has been dropped from most of the major commercial LP codes, we felt that some evaluation of PNET/LP on GUB problems would be of interest to practitioners. Our test problems were furnished by a major airplane manufacturer. The GUB portion represents the assignment of plane-types to routes. The typical problem contained 80 GUB rows, 14 non-network rows, and 130 arc variables. On these small problems PNET/LP was four times faster than MPSX/370.

The third class of problems is the one which PNET/LP is specifically designed to solve. Most of our computational experience with this problem

class is for real problems which were solved by Agrico Chemical Company. These problems involve the determination of optimal production and distribution schedules, and were solved on an AMDAHL V-6 using a FORTRAN G compiler. Unfortunately, it was not possible to compare PNET/LP on these problems to another code because of the proprietary nature of the problem data. (In addition, it is difficult to obtain free computer time on an IBM 370/168 or an AMDAHL V-6 for the purpose of benchmarking against MPSX, MPSX-370, or MPS-III.) Table I contains typical solution statistics on Agrico's three largest product models.

Subsequent to this testing, Agrico acquired a FORTRAN H compiler. Agrico has determined that the FORTRAN H compiler reduces total run time, including all I/O by approximately 45%. Consequently, the times in Table I would probably be reduced by 45% using the FORTRAN H compiler. Furthermore, Agrico's comparison of PNET-I and PNET/LP using the H compiler indicates that PNET/LP is only 20% slower than PNET-I.

TABLE I

SOLUTION STATISTICS ON PNET/LP

No. of Constraints		No. of Variables		PNET/LP*	
No. of Network Rows ≡ m	No. of Non-Network Rows ≡ q	Arcs ≡ n	Non-Arcs ≡ p	Total Time**	Total Pivots
3179	20	15,831	40	103 seconds	10,248
3442	6	21,898	12	180 seconds	17,817
6192	10	21,939	20	351 seconds	10,345

* AMDAHL V-6 with FORTRAN G compiler

** Including all I/O processing

In order to provide some comparison of PNET/LP to a commercial LP code on the third class of problems, we solved one randomly generated problem on the CYBER-74 computer using PNET/LP and APEX-III. Table II contains the problem specifications and indicates that this problem can be solved at least 70 times less expensively with PNET/LP than with APEX-III.

TABLE II

PNET/LP VS. APEX-III

No. of Constraints		No. of Variables		PNET/LP	APEX-III
No. of Network Rows ≡ m	No. of Non-Network Rows ≡ q	Arcs ≡ n	Non-Arcs ≡ p		
1000	1	5000	1	\$3.11*	\$210.68**

*AMDAHL V-6 with FORTRAN G compiler

** Including all I/O processing

While the above computational testing is quite limited, it indicates that the simplex SON algorithm may be extremely efficient for solving large embedded network/LP problems. At this point, however, we would caution the reader that it would be premature to extrapolate these results to other problems and in particular, to other problem classes. An exhaustive computational study is required before any general inferences can be drawn.

ACKNOWLEDGMENTS

We wish to acknowledge many helpful conversations with Harvey Greenberg during the development of the simplex SON procedure, particularly dealing with potential applications. We are also grateful to Leon Lasdon, John Mulvey, and Richard O'Neill for their valuable editorial comments on an earlier version of this paper. Furthermore, we would like to thank David Karney, who did the major part of the implementation and computational testing reported in the final section.

REFERENCES

1. A. Ali, R. Helgason, J. Kennington, and H. Lall, "Solving Multicommodity Network Flow Problems," Technical Report IEOR 77015, Southern Methodist University, 1977.

2. Analysis, Research, and Computation, Inc., "Development and Computational Testing on a Capacitated Primal Simplex Transshipment Code," ARC Technical Research Report, Austin, Texas.

3. A. A. Assad, "Multicommodity Network Flows--Computational Experience," Working Paper No. OR 058-76, Massachusetts Institute of Technology (1976).

4. M. D. Bakes, "Solution for Special Linear Programming Problems with Additional Constraints," *Operational Research Quarterly*, 17, 4 (1966) 425-445.

5. R. Barr, F. Glover, and D. Klingman, "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," *INFOR*, 17, 1 (1979).

6. R. Barr, F. Glover, and D. Klingman, "The Alternating Basis Algorithm for Assignment Problems," *Mathematical Programming*, 13 (1977) 1-13.

7. J. M. Bennett, "An Approach to Some Structured Linear Programming Problems," *Operations Research*, 14 (1966) 636-645.

8. G. Bradley, G. Brown, and G. Graves, "Design and Implementation of Large-Scale Primal Transshipment Algorithms," *Management Science* 24 (1977) 1-35.

9. A. Charnes and W. Cooper, *Management Models and Industrial Applications of Linear Programming*, Vols. I and II, Wiley, New York, 1961.

10. A. Charnes, F. Glover, D. Karney, D. Klingman, and J. Stutz, "Past, Present and Future of Large-Scale Transshipment Computer Codes and Applications," *Computers and Operations Research*, Vol. 2 (1975) 71-81.

11. H. Chen and C. G. DeWald, "A Generalized Chain Labeling Algorithm for Solving Multicommodity Flow Problems," *Computers and Operations Research*, 1 (1974) 437-465.

12. J. E. Cremeans, R. A. Smith, and G. R. Tyndall, "Optimal Multicommodity Network Flows with Resource Allocation," *Naval Research Logistics Quarterly*, 17 (1970) 269-280.

13. J. E. Cremeans and H. S. Weigel, "The Multicommodity Network Flow Model Revised to Include Vehicle Per Time Period and Node Constraints," *Naval Research Logistics Quarterly*, 19 (1972) 77-89.

14. H. Crowder, M. Held, and P. Wolfe, "Validation of Subgradient Optimization," *Mathematical Programming*, 6 (1974) 62-88.

14a. G. B. Dantzig, "Upper Bounds, Secondary Constraints, and Block Triangularity in Linear Programming," *Econometrica*, 23 (1955) 174-183.

15. G. B. Dantzig and R. M. Van Slyke, "Generalized Upper Bounding Techniques," *Journal of Computer and System Science*, 1, 3 (1967) 213-226.

16. L. R. Ford and D. R. Fulkerson, "A Suggested Computation for Maximal Multi-Commodity Network Flows," *Management Science*, 5, 1 (1958) 97-101.

17. J. Gilsinn and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," NBS Technical Note 772, U.S. Department of Commerce, 1973.

18. F. Glover, J. Hultz, and D. Klingman, "Improved Computer-Based Planning Techniques," *Interfaces*, 8, 4 (1978) 16-25.

19. F. Glover, D. Karney, and D. Klingman, "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code," *Networks*, 4, 3 (1974) 191-212.

20. F. Glover, D. Karney, D. Klingman, and R. Russell, "Solving Singly Constrained Transshipment Problems," *Transportation Science*, 12, 4 (1978).

21. F. Glover and D. Klingman, "Capsule View of Future Developments on Large-Scale Network and Network-Related Problems," Research Report CCS 238, Center for Cybernetic Studies, The University of Texas at Austin (1975).

22. F. Glover and D. Klingman, "Finding Minimum Weight Node Order Constrained Spanning Trees," In *Combinatorial Programming: Methods and Applications*, B. Roy (ed.), D. Reidel Publishing Co., Dordrecht-Holland, 1975, 60-71.

23. G. W. Graves and R. D. McBride, "The Factorization Approach to Large-Scale Linear Programming," *Mathematical Programming*, 10, 1 (1976) 91-111.

24. H. J. Greenberg and R. P. O'Neill, "A Computational Perspective of PIES," Supply and Integration Analysis Division, Federal Energy Administration (1977).

25. M. D. Grigoriadis and K. Ritter, "A Decomposition Method for Structured Linear and Nonlinear Programs," *Journal of Computer and System Sciences*, 3 (1969) 335-360.

26. J. K. Hartman and L. S. Lasdon, "A Generalized Upper Bounding Method for Doubly Coupled Linear Programs," *Naval Research Logistics Quarterly*, 17, 4 (1970) 411-429.

27. J. K. Hartman and L. S. Lasdon, "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems," *Networks*, 1 (1972) 333-354.

28. A. R. G. Heesterman, "Special Simplex Algorithm for Multisector Problems," *Numerische Mathematik*, 12 (1968) 288-306.

29. R. Helgason and J. Kennington, "A Product Form Representation of the Inverse of a Multicommodity Cycle Matrix," Technical Report IEOR 76003, Southern Methodist University (1976).

30. R. Helgason, J. Kennington, and J. Lall, "Primal Simplex Network Codes: State-of-the-Art Implementation Technology," Technical Report IEOR 76014, Department of Industrial Engineering and Operations Research, Southern Methodist University (1976).

31. J. Hultz and D. Klingman, "Solving Singularly Constrained Generalized Network Problems," *Applied Mathematics and Optimization*, 4 (1978) 103-119.

32. J. Hultz and D. Klingman, "Solving Constrained Generalized Network Problems," Research Report CCS 257, Center for Cybernetic Studies, The University of Texas at Austin (1976).

33. R. N. Kaul, "An Extension of Generlaized Upper Bounded Techniques for Linear Programming," ORC 65-27, University of California, Berkeley (1965).

34. J. Kennington and M. Shalaby, "An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems," Technical Report IEOR 75010, Department of Industrial Engineering and Operations Research, Southern Methodist University (1976).

35. D. Klingman and R. Russell, "On Solving Constrained Transportation Problems," *Operations Research*, 23, 1 (1975) 91-107.

36. S. F. Maier, "A Compact Inverse Scheme Applied to a Multicommodity Network with Resource Constraints," Technical Report No. 71-8, Operations Research House, Stanford Univeristy (1971).

37. R. D. McBride, "Factorization in Large-Scale Linear Programming," Working Paper No. 22, University of California, Los Angeles (1973).

37. R. D. McBride, "Factorization in Large-Scale Linear Programming," Working Paper No. 200, University of California, Los Angeles (1973).

38. R. Saigal, "Multicommodity Flows in Directed Networks," ORC Report 66-24, Operations Research Center, University of California, Berkeley (1966).

39. R. Saigal, "Multicommodity Flows in Directed Networks," Technical Report No. ORC 67-38, Operations Research Center, University of California, Berkeley (1967).

40. M. Sakarovitch, "The Multi-Commodity Maximum Flow Problem," Technical Report No. ORC 66-25, Operations Research Center, University of California, Berkeley (1966).

41. V. Srinivasan and G. Thompson, "Accelerated Algorithms for Labeling and Relabeling of Trees with Applications for Distribution Problems," *JACM*, 19, 4 (1972) 712-726.

42. C. Swoveland, "Decomposition Algorithms for the Multi-Commodity Distribution Problem," Working Paper No. 184, Western Management Science Institute, University of California, Los Angeles (1971).

43. C. Swoveland, "A Two-Stage Decomposition Algorithm for a Generalized Multi-Commodity Flow Problem," *INFOR*, 11, 3 (1973) 232-244.

44. J. A. Tomlin, "Minimum-Cost Multicommodity Network Flows," *Operations Research*, 14, 1 (1966) 45-51.

45. J. A. Tomlin, "Mathematical Programming Models for Traffic Network Problems," Unpublished Dissertation, Department of Mathematics, University of Adelaide, Australia (1967).

46. D. W. Webber and W. W. White, "An Algorithm for Solving Large Structured Linear Programming Problems," IBM New York Scientific Center Report No. 320-2946 (1968).

47. R. D. Wollmer, "Multicommodity Networks with Resource Constraints: The Generalized Multicommodity Flow Problem," *Networks*, 1 (1972) 245-263.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Center for Cybernetic Studies
The University of Texas at Austin

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE (6)
The Simplex SON Algorithm for LP/Embedded Network Problems (9) Research rept.

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name) (10)
Fred Glover
Darwin Klingman

(11) Jan 79 (12) 52p.

6. REPORT DATE (15)
Revised January 1979

7a. TOTAL NO. OF PAGES
46

7b. NO. OF REFS
47

8a. CONTRACT OR GRANT NO.
N00014-75-C-0616, 0569
ERA-Cr-03-70128-00 N00014-75-C-0569

8b.

8c. NR 047-021 and NR 047-172

8d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Center for Cybernetic Studies
Research Report CCS-317 (14)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Office of Naval Research (Code 434)
Washington, D.C.

13. ABSTRACT This paper develops a special partitioning method for solving LP problems with embedded network structure. These problems include many of the large-scale LP problems of practical importance, particularly in the fields of energy, scheduling, and distribution. The special partitioning method, called the simplex special ordered network (SON) procedure, applies to LP problems that contain both non-network rows and non-network columns, with no restriction on the form of the rows and columns that do not exhibit a network structure. These LP/embedded network problems include as a special case multi-commodity network problems and constrained network problems previously treated in the literature, by simultaneously encompassing both side constraints and side activities.

The simplex SON procedure solves these problems by exploiting the network topology of the basis, whose properties are characterized by means of a specially constructed *master basis tree*. A set of fundamental exchange rules are developed which identify admissible ways to modify the master basis tree, and hence the composition of the partitioned basis inverse. The simplex SON method implements these rules by a set of streamlined labeling algorithms, while maintaining the network portion of the basis as large as possible, thereby accelerating the basis exchange step. As a result, LP/embedded network problems can be solved with less computer storage and significantly greater efficiency than available from standard linear programming methods.

Preliminary computational results are reported for an all-FORTRAN implementation of the simplex/SON algorithm called PNET/LP. The test problems are real-world models of physical distribution and scheduling systems. PNET/LP has solved problems with 6200 rows and 22,000 columns in less than 6 minutes, counting all I/O, on an AMDAHL V-6 with a FORTRAN G complier.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

406 197

Unclassified
Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Linear Programming						
Networks						
Graphs						
Energy Model						
Transportation						
Distribution						